

APPLICATION
FOR
UNITED STATES LETTERS PATENT

TITLE: FORMATTING CONTENT BY EXAMPLE

APPLICANT: H.S. ROY AND JEFF YOUNG

Express Mail Label No. EL444262465US

December 7, 1999
Date of Deposit

09456888-120799
662027-88895460

FORMATTING CONTENT BY EXAMPLE

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights.

BACKGROUND OF THE INVENTION

The invention relates to apparatus and process for generating content.

The Internet is becoming an increasingly popular medium for communicating and publishing widely accessible documents. A network of networks, the Internet transfers information using a common protocol that tells computers connected to the network how to locate and exchange files with one another. Documents communicated over the Internet generally conform to a Hyper Text Markup Language (HTML) that a World-Wide-Web (WWW or Web) browser can translate and display. Once posted on a Web server, these documents or compilations of pages can be retrieved and viewed by anyone who has access to the Internet.

Each document is essentially a collection of HTML codes or tags which provide the document with a general structure of a head and a body, as well as headings, format controls, forms, tables, and one or more element types. A basic familiarity with HTML codes is only one aspect in the process of creating HTML documents. Another issue relates to the process of editing such documents. Although a conventional text editor or word processor can be

used to add HTML markups to the document, such method of composing and editing the HTML document is quite tedious as the process does not allow a designer to see the document as actually displayed by the browser. Without visual feedback, the process of composing and editing the HTML document can be error-prone and inefficient. Moreover, when the HTML document contains elements other than the usual text and text formatting codes, the process of composing and editing the HTML document can be challenging. For example, if the image elements were embedded in the document, the conventional text editor or word processor would reference each image using only its access path and file name. Consequently, the designer has to be more careful and more exact in selecting the elements, as the designer cannot visually verify that the correct image is being edited. Hence, the difficulty in generating the desired HTML document is increased when non-text elements are embedded in the document.

The difficulty is particularly accentuated for the development of web sites in which the content or interaction of the web site changes based on designer or environmental information. Examples of such sites include on-line catalogs, news sites, or e-commerce sites.

Typically, to support dynamic web sites, a web server called a dynamic content server processes pages to insert or replace content at run-time. Examples include Microsoft's Internet Information Server (IIS), which processes Active Server Pages (ASP).

Currently, web sites are generally authored in two steps:

(1) A designer creates the look and feel of the web site by creating a mock-up of the site in Photoshop.

(2) The Photoshop image is then handed to programmers who hand-code the web pages in HTML and script code to be executed by a content server.

This two-step process is caused by the fact that WYSIWYG HTML editors are insufficiently expressive to show what the designer wants, and; that existing content specification systems generally imbed processing instructions directly within the HTML, making it hard for them to co-exist with WYSIWYG HTML editors. An example of these processing instructions is exemplified by the following Active Server Page (ASP) code

```
<BODY>
<%
set patents = CreateObject("ADODB.RecordSet")
patents.Open "myPatentDatabase"
%>
<TABLE>
<TR>
<TH>Title</TH>
<TH>Description</TH>
</TR>

<% while not patents.EOF %>
<TR>
<TD><%=patents("PATENT_TITLE")%></TD>
<TD><%=patents("SHORT_DESCRIPTION")%></TD>
</TR>
<% patents.MoveNext %>
```

This two-step process is inefficient since the page is in effect designed twice: once in Photoshop, and once in HTML file. Moreover, the designer cannot go back and make changes to the HTML file, because the ASP code cannot be easily read back into a WYSIWYG editor since ASP is a language-independent framework designed by Microsoft for efficient coding of server-side scripts that are executed by a Web server in response to a user's request for a URL. ASP scripts are similar to other server-side scripting such as Perl and Python. Not surprisingly, although many people can design web pages, only a few designers can compose web sites with ease.

SUMMARY OF THE INVENTION

The system and techniques specify content insertions based on elements, each with a representative content. The format, style, and layout of the representative content of the elements is used to format, style, and layout the content.

5 A method generates a document by: inserting one or more placeholders in the document; binding each placeholder to a content source; and compiling the placeholders into code to generate content for the document based on the content source. The method then executes the code to generate the content and presenting the content of the document if in a presentation mode or restores the placeholder if in an edit mode.

10 Implementations of the invention may include one or more of the following. The placeholders can be formatted and annotated. Executable code can then be generated from the annotations to retrieve the content. The document can then be rendered based on the content. The generating of executable codes can include generating code to obtain content from a database. The code can be embedded as attributes in the placeholder. The placeholder can be a mark-up element, and can be an HTML element or an XML element. The method can include specifying a content insertion based on the placeholder, each placeholder having a representative content. The method can also use the formatting, styling, or layout of the representative content of the placeholder to format, style, or layout the content.

20 In another aspect, a method generates a document by inserting one or more placeholders in the document; binding each placeholder to a content source; and interpreting the placeholders and presenting the content of the document if in a presentation mode or restoring the placeholder if in an edit mode.

Implementations of the invention may include one or more of the following. Each placeholder includes one or more annotations describing the content for the placeholder. These annotations are used to generate content to replace the placeholder. The content as interpreted by the annotations are retrieved from the content sources. Each placeholder can be formatted. The formatting the content can be adjusted to match the format of the placeholder. The method can also interpret the layout; and replace the content in lieu of the placeholder when rendering the document. The placeholder can be restored when the designer edits the page in an edit mode. Metadata for the placeholder can be stored as a comment field in the document.

In another aspect, a method applies a document by: inserting one or more mark-up elements in the document; binding the mark-up elements to one or more content sources; retrieving data from the bound one or more content sources and replacing the mark-up elements with the retrieved data during run-time; and restoring the mark-up elements during an edit session.

Implementation of this method can include updating a markup language document during run-time based on an original layout and content generated on-the-fly. Further, the content replaces the placeholder.

In another aspect, a method generates a document having one or more mark-up elements, each elements being bound to a content source. This is done by specifying one or more parameters; retrieving data satisfying the one or more parameters from the bound content source; replacing the mark-up elements with the retrieved data; and displaying the document.

Advantages of the system may include one or more of the following. A designer can use a WYSIWYG editor to create a web site design in HTML. The system then uses the

5

20

5

20

20

20

containing the latest multimedia features, without requiring any HTML programming. At the same time, it provides HTML source-code and JavaScript tools to help Web designers and programmers integrate interactivity into their sites.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a flowchart illustrating a process for generating content using one or more mock placeholders.

Fig. 2A is an exemplary user interface illustrating one or more mock placeholders and their content sources.

Fig. 2B is an exemplary user interface illustrating one or more mock placeholders and binding details associated with the placeholders.

Fig. 3 is a flowchart of a process for processing the mock placeholders.

Fig. 4 is an exemplary user interface illustrating various exemplary mock placeholders and their content sources.

Fig. 5 is a flowchart of a process for interpreting the content associated with the mock placeholders.

Fig. 6 is an exemplary computer system for executing the process of Fig. 1.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to Figure 1, a process 100 for editing and running pages with content is shown. In applying the process 100, a designer creates pages as usual, placing mock elements or placeholders on the page in place of actual content. The mock content lets the designer visually design the page exactly the way it should appear to the designer. After designing the page in this way, the designer binds or links the placeholders to fields in an online database, or a server such as an e-commerce server that provides data. The content of those fields replaces the mock content on the page when it appears live or is previewed.

Specifically, the designer layouts a page with mock content placeholders (step 102). Next, the designer formats the mock content placeholders as necessary (step 104). The

5

10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995
1000
1005
1010
1015
1020
1025
1030
1035
1040
1045
1050
1055
1060
1065
1070
1075
1080
1085
1090
1095
1100
1105
1110
1115
1120
1125
1130
1135
1140
1145
1150
1155
1160
1165
1170
1175
1180
1185
1190
1195
1200
1205
1210
1215
1220
1225
1230
1235
1240
1245
1250
1255
1260
1265
1270
1275
1280
1285
1290
1295
1300
1305
1310
1315
1320
1325
1330
1335
1340
1345
1350
1355
1360
1365
1370
1375
1380
1385
1390
1395
1400
1405
1410
1415
1420
1425
1430
1435
1440
1445
1450
1455
1460
1465
1470
1475
1480
1485
1490
1495
1500
1505
1510
1515
1520
1525
1530
1535
1540
1545
1550
1555
1560
1565
1570
1575
1580
1585
1590
1595
1600
1605
1610
1615
1620
1625
1630
1635
1640
1645
1650
1655
1660
1665
1670
1675
1680
1685
1690
1695
1700
1705
1710
1715
1720
1725
1730
1735
1740
1745
1750
1755
1760
1765
1770
1775
1780
1785
1790
1795
1800
1805
1810
1815
1820
1825
1830
1835
1840
1845
1850
1855
1860
1865
1870
1875
1880
1885
1890
1895
1900
1905
1910
1915
1920
1925
1930
1935
1940
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
2030
2035
2040
2045
2050
2055
2060
2065
2070
2075
2080
2085
2090
2095
2100
2105
2110
2115
2120
2125
2130
2135
2140
2145
2150
2155
2160
2165
2170
2175
2180
2185
2190
2195
2200
2205
2210
2215
2220
2225
2230
2235
2240
2245
2250
2255
2260
2265
2270
2275
2280
2285
2290
2295
2300
2305
2310
2315
2320
2325
2330
2335
2340
2345
2350
2355
2360
2365
2370
2375
2380
2385
2390
2395
2400
2405
2410
2415
2420
2425
2430
2435
2440
2445
2450
2455
2460
2465
2470
2475
2480
2485
2490
2495
2500
2505
2510
2515
2520
2525
2530
2535
2540
2545
2550
2555
2560
2565
2570
2575
2580
2585
2590
2595
2600
2605
2610
2615
2620
2625
2630
2635
2640
2645
2650
2655
2660
2665
2670
2675
2680
2685
2690
2695
2700
2705
2710
2715
2720
2725
2730
2735
2740
2745
2750
2755
2760
2765
2770
2775
2780
2785
2790
2795
2800
2805
2810
2815
2820
2825
2830
2835
2840
2845
2850
2855
2860
2865
2870
2875
2880
2885
2890
2895
2900
2905
2910
2915
2920
2925
2930
2935
2940
2945
2950
2955
2960
2965
2970
2975
2980
2985
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
36

20

1. Find the tag that corresponds to the selected object.
2. Find the attribute list for the tag
3. Compute the annotations needed to describe the behavior of the selected object.
4. Add the annotations as attributes to the attribute list, replacing old attributes that have the same name with the new ones.

From step 106, the annotations are used to compile the placeholders into executable code to retrieve content (step 108). This is done when the page is saved as a file.

5 Next, in step 110, the process 100 determines whether the user is in edit mode or view mode. If the user is in the view mode, the process 100 proceeds to step 112 where the page is displayed. In this step, each placeholder is analyzed, and based on the type and the annotations of the placeholder, appropriate code is generated that will put in the actual content. Alternatively, the annotations can be interpreted to fill the page with content. At run-time or edit time, the information in the annotations is processed to replace the placeholders with the actual content (interpretation). Finally, the page can be rendered with the content.

From step 110, if the user is in the edit mode, when the page is read, the code is decompiled back to the original placeholders and annotations so that the designer can edit them and change the layout (step 114). From step 112 or 114, the process 100 exits.

Figure 2A shows an exemplary layout using an HTML editor such as Adobe's GoLive. GoLive provides a context-sensitive palette called the Inspector can be used to let the designer quickly customize objects without using commands in the menu bar. The context-sensitive Inspector can be used to set the attributes of text or objects in the page or site window. Inspectors are the source of point-and-shoot linking; they also let the designer reference image files or other resources, resize elements with pixel-level precision, align objects with respect to adjacent text, control object spacing between objects, and make many tag-specific settings.

20 The layout 120 of Figure 2A has a background table, which in turn includes table cells 124, 126, and 140. The table cell 124 contains various elements such as of an "Adobe Magazine" icon and a "Contents" table. The table cell 126 contains text fields "Ducky takes 1st prize" and "The ImageReady rubber ducky has surprised everyone, including its creators, by taking first prize in a local duck beauty pageant." The table cell 126 also includes an

5

10 11 12 13 14 15 16

20

Operations associated with an exemplary placeholder are discussed next. In Figure

then linking the placeholder to an image file. The image placeholder can then appropriately positioned on the page.

Similarly, to insert a text element with a title and a body, two text placeholders 132 and 134 on the page are created. The box next to a Title field listed in the content sources pane 150 is selected to bind the placeholder 132 to the Title field of the database. Next, the body placeholder 134 is selected, and the box next to the Introduction field in the Content Sources pane 150 is selected to bind the placeholder 134 to another field in the database. Finally, the Show Bindings button in the Dynamic Content palette is selected. At this point, the page can be previewed to see the content in action.

Specific portions of the HTML code for the sample page shown in Fig. 2A is shown below:

```
<!-- #INCLUDE FILE="../config/include/utils.asp" -->
<!-- #INCLUDE FILE="../config/include/ado.asp" -->
<!DOCTYPE html PUBLIC "-//W3C/DTD HTML 4.0 Transitional/EN">
<html>

<head>
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
  <meta name="generator" content="Adobe GoLive 4">
  <title>Adobe Magazine</title>
  <%Dynamic Link
    set Feature_InformationRecordSet = CreateObject("ADODB.RecordSet")
    Feature_InformationRecordSet.Open "select * from features order by Priority",
    ConnectString("Magazine")
    set Feature_Information = WrapRecordSet(Feature_InformationRecordSet)
  %>
</head>

<body bgcolor="white">
  <table border="0" cellpadding="0" cellspacing="10">
    <tr height="500">
      <td valign="top" height="500">
        <h2></h2>
        <h2>
          <table border="0" cellpadding="0" cellspacing="0" width="145">
            <tr>
              <td width="33"></td>
```

```

        <td>
        <h2>Contents</h2>
        <p><a href="frontpage.asp">Home</a><br>
        Features<br>
5      Q & A</td>
        </tr>
        </table>
        </h2>
        </td>
10     <td valign="top" align="right" height="500">
        <h2><%=Feature_Information.Value("Title")%></h2>
        <p><%=Feature_Information.Value("Introduction")%></p>
        <p><img src='<%=Feature_Information.Value("ImageURL")%>' width="280"
height="170"></p>
15     <p><a
href="fullarticle.asp?RECORD_KEY=$queryString&<%=Feature_Information.Key()%>">f
ull story</a></td>
        <td width="1" valign="top" bgcolor="black" height="500"></td>
        <td valign="top" height="500">
        <h2>Features</h2>
        <p>
        <table>
        <% Feature_Information.MoveFirst %>
        <% Feature_Information.Move 2 - 1 %>
        <% while not Feature_Information.EOF %>
        <tr>
        <td><a
20 href="fullarticle.asp?RECORD_KEY=$queryString&<%=Feature_Information.Key()%>"><
%=Feature_Information.Value("Title")%></a>
        <hr>
        </td>
        </tr>
        <% Feature_Information.MoveNext %>
35     <% wend %>
        </table>
        <p>
        <p>more stories...</td>
        </tr>
        </table>
        </body>

</html>
<% if 0 then %>
45 <!--Mock Content
    Feature_Information:<TABLE> {<tr> <td><a href="fullarticle.asp">Kidnappers foiled in
plot to nab ducky <hr> </a></td> </tr> <tr> <td><a href="fullarticle.asp">Geese
making angry noises about rigged judging <hr> </a></td> </tr> <tr> <td><a
```

5

10

15

20

30

database (step 202). In one embodiment, code to get information from a database is as follows:

```
<%  
    set Story_InformationRS= CreateObject("ADODB.RecordSet")  
    Story_InformationRS.Open "select * from features",  
    ConnectString("Magazine")  
    set Story_Information= WrapRecordSet(Story_InformationRS)  
    Story_Information.Move Request.QueryString("RECORD_INDEX") - 1  
%>
```

Next, this code is embedded as HTML or XML attributes on the placeholder (step 204) or simply stored as an HTML element or an XML element. XML is a simplified dialect of SGML (Standard Generalized Markup Language) for the structured presentation of information on the World Wide Web. XML is a subset of SGML for defining custom markup languages. XML documents use Document Type Definitions (DTDs) that define the custom tags available for use in the document. More information on XML is provided at www.w3.org/XML/ or at www.xml.com/

Finally, the process 200 also saves mock content placeholders so that, it should be used as decide to edit the page in a edit mode, the mock content placeholders can be restored for the designer to edit.

Although the process 200 shows content generation from the point of a designer, a user can influence the generation of content as well. In another embodiment, the user can control the type of content that the web page is to display. The user can specify his or her desired content using a qualifier along with the entry of a web site address. For example, the user can search for all stories by a particular author by entering "www.publisher.com author = Tran." The returned web page would be customized to the user's qualifier and would return only contents that had been authored by the desired user qualifier. The code generated for a link of this example is shown below:

5 <%Dynamic Link set Story_InformationRecordSet =
CreateObject("ADODB.RecordSet")
Story_InformationRecordSet.Open "select * from Features where Author =
" & Request.QueryString("Author") & " order by Priority",
ConnectString("Magazine")
set Story_Information = WrapRecordSet(STory_InformationRecordSet)
%>

10 To illustrate with an example, a placeholder element with annotations typically is
expressed in HTML as:

15 The IMG tag additionally has an attribute called dynamic_src, which represents an
annotation, which allows the system of Fig. 1 to turn the mock placeholder into content
during a "live" rendering of the page. In addition, in edit mode, an editor such as Adobe
GoLive can still show the IMG tag as a regular image of a duck.

In one embodiment, code to generate an image is:

20 <%
[code to get content from the database]
%>
...
<IMG src="<%=Story_Information.Value("ImageUrl")%>">
...
<!-- Mock Content Story_Information:ImageUrl {ducky.jpg} -->

25 The last section lets the system reverse the compilation process for generating pages
and turn the code back into an HTML page that the designer can edit the page as necessary.

30 The invention operates with various other placeholder types. For instance, the page
layout of Figure 4 has a text placeholder 250, an image placeholder 252, a table placeholder
254, a text input box placeholder 256, a pull-down menu placeholder 258, a radio box
placeholder 260, a checkbox placeholder 262 and a hotlink placeholder 264. A content
source tab 270 is in turn associated with each of the placeholders 250-264.

Exemplary code for the text placeholder 250 is as follows:

```
<%=Product_Info.Value("Name")%>
```

This code inserts the text of the "Name" field from the current record of the "Product_Info" table. The current record is set by the code that retrieves the database information. The formatting depends on the tags surrounding the code. If the code is within an <H1> tag, for example, it is formatted as a heading.

Exemplary code for the image placeholder 252 is as follows:

```
<img height="32" width="32" src='<%=Information.Value("ImageURL")%>'
```

This code replaces the src attribute of the image with the text of the ImageURL field. Replacing the src attribute causes the user's browser to retrieve the image referenced by the ImageURL.

Exemplary code for the table placeholder 254 is as follows:

```
<table border="4" cellpadding="0" cellspacing="2" width="119">  
<% Product_Info.MoveFirst %>  
<% while not Product_Info.EOF %>  
<tr>  
<td><%=Product_Info.Value("Title")%></td>  
</tr>  
<% Product_Info.MoveNext %>  
<% wend %>  
</table>
```

This code constructs a table that shows all the records in the Product_Info table, with one record per row. Each row will display the Name of one product. The table will have the formatting indicated by the attributes in the <TABLE> tag.

Exemplary code for the text input box placeholder 256 is as follows:

```
<input type="text" value='<%=Product_Info.Value("Name")%>'  
name="Title" size="24">
```

This code creates a text input box for updating the Name field of the current record.

The information will be uploaded to the database as part of a form submission.

Exemplary code for the pull-down menu placeholder 258 is as follows:

```
<select name="Title" size="1">
<% Shipping_Methods.MoveFirst %>
<% while not Shipping_Methods.EOF %>
<option value='<%=Shipping_Methods.Value("MethodName")%>'
name="ShippingMethod"
<%=selected(Product_Info.Value("ShippingMethod")
Shipping_Methods.Value("MethodName"))%>><%=Shipping_Methods.Value("Met
hodName")%>
<% Shipping_Methods.MoveNext %>
<% wend %>
</select>
```

This code constructs a pull down menu to update the ShippingMethod field of the current record. The entries in the menu are created from the records in the Shipping_Methods table, each of which has a MethodName. The code is further designed to show the shipping method assigned to the current record.

Exemplary code for the radio box placeholder 260 is as follows:

```
<input type="radio" value='<%=Information.Value("Subtitle")%>'
name="Title" <%=checked(Information.Value("Title")
Information.Value("Subtitle"))%>> <%=Information.Value("Title")%>

<table border="0" cellpadding="2" cellspacing="2">
<% Status_List.MoveFirst %>
<% while not Status_List.EOF %>
<tr>
<td><input type="radio" value='<%=Status_List.Value("ProjectStatus")%>'
name="ProjectStatus" <%=checked(Project_Information.Value("ProjectStatus")
Status_List.Value("ProjectStatus"))%>><%=Status_List.Value("ProjectStatus")%>
</td>
</tr>
<% Status_List.MoveNext %>
<% wend %>
</table>
```

5

10

15

20

25

The design is created using existing HTML elements. An element can be tagged (using an attribute, among others) to be associated with some form of content. At run-time, the representative content is examined to format, style, and layout the content.

5 The annotations are stored as XML or HTML attributes. This has the advantage that the file stays an XML or an HTML file, so that the designer can continue to edit it using existing XML and HTML tools. In contrast, other output formats do not use this model. ASP code, for example, is unstructured and cannot typically be read into an HTML editor.

10 Using the attribute format makes it easy to apply an interpreter that processes the content at runtime. An interpreter is a computer program that translates and executes instructions written in an interpreted programming language, such as programs written in Perl, Basic, or Java. The translation process involves the conversion of each source language statement of a given computer program into machine language suitable for execution. A user wishing to execute an application program written in the programming language will invoke an interpreter in order to execute the application program. The interpreter can simply parse the page as an XML or HTML document, and then using the standard DOM interface for accessing such a document, it can find the content annotations, fetch the required content, and substitute for the placeholders.

20 Referring now to Fig. 5, a process 300 for interpreting the content is shown. First, the process 300 retrieves the next placeholder from the document (step 302). The process 300 then reads the annotations describing the content that should replace that placeholder (step 304). The required content is retrieved (step 306). The process 300 then formats the content to match the formatting of the placeholder (step 308) and replaces the placeholder with the content (step 310). Next, the process 300 checks whether all placeholders have been processed (step 312). If not, the process 300 loops back to step 402 to continue processing

the placeholders. Alternatively, the process 300 proceeds from step 312 to display the final document showing all the content (step 314).

Pseudo-code to display a document by interpreting the content annotations is shown below:

1. Read the next placeholder from the document.
2. Read the annotations that describe the content that should replace that placeholder.
3. Retrieve the required content.
4. Format the content to match the formatting of the placeholder.
5. Replace the placeholder with the content.
6. Continue with step 1 until all placeholders have been processed.
7. Display the final document showing all the content.

The invention may be implemented in digital hardware or computer software, or a combination of both. Preferably, the invention is implemented in a computer program executing in a computer system. Such a computer system may include a processor, a data storage system, at least one input device, and an output device. Figure 6 illustrates one such computer system 600, including a processor (CPU) 610, a RAM 620, a ROM 622 and an I/O controller 630 coupled by a CPU bus 698. The I/O controller 630 is also coupled by an I/O bus 650 to input devices such as a keyboard 660, a mouse 670, and output devices such as a monitor 680. Additionally, one or more data storage devices 692 are connected to the I/O bus via an I/O interface 690.

Further, variations to the basic computer system of Figure 5 are within the scope of the present invention. For example, instead of using a mouse as user input devices, a pressure-sensitive pen, digitizer or tablet may be used.

The above-described software can be implemented in a high level procedural or object-oriented programming language to operate on a dedicated or embedded system. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

Each such computer program can be stored on a storage medium or device (e.g., CD-ROM, hard disk or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described. The system
5 also may be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

Other embodiments are within the scope of the following claims.

05456888-120799